

# THE UNIVERSITY OF CHICAGO

1  
2 **TECHNICAL FIELD**

3 This invention relates generally to automotive computing devices.

4  
5 **BACKGROUND**

6 Automobiles are becoming increasingly popular platforms on which to  
7 provide computing devices. One popular computing device in the automotive  
8 space is Microsoft's Auto PC. Powered by the Microsoft Windows CE operating  
9 system, the Auto PC is Microsoft's in-car entertainment and information platform  
10 technology. Hardware versions of the Auto PC platform can fit into most  
11 automobile dashboards, have color LCD screens, high-powered AM/FM stereos,  
12 and CD-ROM drives. The inclusion of the CD-ROM drive allows users to access  
13 vast stores of data on their Auto PC. The Auto PC is ideally an extensible platform  
14 which can be built upon to provide added applications and functionality for the  
15 user. For example, applications can be provided that enable drivers to use voice  
16 commands to check e-mail and schedules, find phone numbers, make calls on their  
17 car phones and get news and other information. The Auto PC can include  
18 applications that permit wireless Internet access for the purpose of searching and  
19 retrieving information over the Web. The Auto PC platform provides a platform  
20 for a seemingly endless number of user applications that can greatly enhance the  
21 user's experience.

22 There are challenges, however, associated with providing computing  
23 devices such as the Auto PC in automobiles.

24 In automobiles using Windows CE or any other type of operating system,  
25 there is typically critical data that the system uses which is usually stored in so-

1 called working RAM (i.e. volatile dynamic random access memory) for speed of  
2 access. This critical data can include application data (e.g. navigation data,  
3 address book data, third party installed applications and the like). Dynamic  
4 random access memory has to be constantly refreshed or the data that it contains  
5 will be lost. If the power is lost, the DRAM cannot be refreshed and, hence, any  
6 critical data that it contains is lost.

7 Current Windows CE implementations, as well as other implementations,  
8 use a small cell back-up battery that is used to back up the entire DRAM when the  
9 system goes in a standby mode associated with, for example, a power loss. For  
10 automotive requirements (such as Auto PC and the like), there are a number of  
11 problems with this solution. Typically, the batteries that are used for backup  
12 purposes do not meet automotive specifications. For example, the temperature  
13 ranges at which the batteries work are not suitable for the temperature ranges  
14 typically encountered by automobiles. Additionally, automotive manufacturers or  
15 original equipment manufacturers (OEMs) do not want consumers to have to go to  
16 the burden of replacing the backup batteries when they eventually fail. The  
17 backup batteries, such as those produced by Tadiran, which do meet automotive  
18 requirements are limited to a few microamps of current draw if the non-  
19 rechargeable battery is to last the life of the product without replacement.

20 Fig. 1 shows a portion of a storage system of a typical automotive computer  
21 generally at 10 that includes non-volatile storage 12 and DRAM 14 coupled  
22 together by a bus 16. Exemplary non-volatile storage can include flash memory or  
23 E<sup>2</sup> memory. This type of memory does not need to be refreshed and can retain its  
24 data in the event of a loss of power. DRAM, as pointed out above, needs to be  
25 constantly refreshed. When the automobile's computer system is booted, its

1 object store (i.e. database, the file system, and the registry) is typically copied  
2 from non-volatile storage 12 into DRAM 14. When the automobile's computer  
3 system enters zero power mode, eliminating any drain on the car's main battery,  
4 the entire contents of DRAM 14 are written back into the non-volatile storage 12.  
5 This ensures that the data is preserved and can be written into DRAM the next  
6 time that the system is booted up. The automobile's computer system may go to a  
7 standby state drawing only a few milliamps (DRAM in self-refresh state) for some  
8 time, perhaps a few days, as determined by the OEMs before entering zero power  
9 mode.

10 This solution works well as long as the system shut down to zero power  
11 mode is an orderly one that is known in advance. Specifically, consider the case  
12 where non-volatile storage 12 is implemented as flash memory. In that case, the  
13 volatile data stored in DRAM (data that needs to be preserved across power  
14 cycles) is written into the flash memory as a *copy* operation. This operation can  
15 take about 90 seconds to complete. In the context of an automotive computing  
16 device, it is often not possible to know ahead of time that the power is going to be  
17 abruptly lost. For example, on a cold day in an automobile with a marginal  
18 battery, when the car is started, the starter can drop the voltage out of regulation  
19 instantaneously. In that case, power loss is instantaneous and any data in DRAM  
20 will be lost.

21 One possible solution to the problem of an abrupt power loss involves using  
22 a capacitor in conjunction with the voltage supply to provide a decay of the supply  
23 voltage in the event of an abrupt power loss. This solution is not optimal or even  
24 desirable because the extra time that is provided for copying data from DRAM to  
25 flash is entirely inadequate. For example, assuming 3A at 12V, by using a 5000

micro Farad electrolytic capacitor to hold the computer system's input voltage, the supply voltage to the ICs can be held for approximately 12 milliseconds before dropping out of regulation. Since many OEMs don't want to use a capacitor this large, the realistic hold time is probably more on the order of 2 to 10ms. This amount of time is not adequate to copy all of the necessary data to non-volatile storage. For example, consider that a single erase and write flash operation for writing to one block of NOR flash memory can take on the order of one second. Each flash block can be on the order of 512 bytes to 4K bytes depending on the flash memory. So within the few milliseconds that are provided by the capacitor solution, at most one Kbyte of data will be able to be written. This is not adequate. Additionally, non-volatile storage such as flash memory requires more power when it is written to. Accordingly, this would further increase the capacitor size requirement.

Another possible solution for maintaining the system's critical data in the event of an abrupt power loss is to continuously write the critical data into the flash memory. There are a number of problems with this approach. First, it takes a long time to write to flash memory (i.e. flash memory has to first be erased and then written to). Additionally, flash memory has a limited read/write capability so that if it is being updated continuously, it is going to wear out. Specifically, flash memory and E<sup>2</sup> memory have on the order of 100,000 – 1,000,000 write cycles per block. By constantly writing to flash memory you are eventually going to wear the device out. Finally, it is possible to corrupt the contents of flash if power is lost during a write to it.

Essentially then, the challenges associated with providing automotive computing devices such as the Auto PC can be distilled down to not having



the event of a power interruption. Additionally, the software can also provide an orderly means by which pages in the SRAM can be written out to flash memory thereby avoiding unnecessary flash write operations which, in turn, increases the lifetime of the flash memory.

## **BRIEF DESCRIPTION OF THE DRAWINGS**

Fig. 1 is a block diagram of an exemplary storage system in an automotive computer.

Fig. 2 is a block diagram of an exemplary automotive computing device that is suitable for use in connection with the described embodiments.

Fig. 3 is a block diagram of a portion of the Fig. 2 computing device.

Fig. 4 is a block diagram of exemplary hardware components in accordance with one described embodiment.

Fig. 5 is a graph or chart that is useful in understanding the operating environment of the described embodiments.

Fig. 6 is a flow diagram that describes steps in a method in accordance with one described embodiment.

Fig. 7 is a block diagram of one circuit in accordance with one described embodiment.

Fig. 8 is a circuit diagram of one circuit in accordance with one described embodiment.

Fig. 9 is a block diagram that illustrates aspects of but one exemplary virtual addressing system.

Fig. 10 is a flow diagram that describes steps in a method in accordance with one described embodiment.

1 Fig. 11 is a diagram that illustrates an exemplary object store page table  
2 entry.

3 Fig. 12 is a table that describes aspects of handling object store page  
4 exceptions.

5 Fig. 13 is a flow diagram that describes steps in a method in accordance  
6 with one described embodiment.

7 Fig. 14 is a diagram that is useful in understanding compaction aspects of  
8 the described embodiments.

9 Fig. 15 is a flow diagram that describes steps in a method in accordance  
10 with one described embodiment.

11 Fig. 16 is a diagram that is useful in understanding wear-leveling aspects of  
12 the described embodiments.

13 Fig. 17 is a flow diagram that describes steps in a method in accordance  
14 with one described embodiment.

## 15 **DETAILED DESCRIPTION**

### 16 **Overview**

17 In accordance with the described embodiments, a cost-effective solution to  
18 data loss problems caused by emergency power shut down in automotive  
19 computing devices is provided in the form of a small amount of static RAM (e.g.  
20 128-256K) that is incorporated into a storage assembly that is utilized by the  
21 computing devices. The storage assembly thus can contain an amount of non-  
22 volatile storage (e.g. flash memory), dynamic random access memory (DRAM or  
23 SDRAM), as well as the SRAM. The SRAM is battery-backed by a small lithium  
24 cell battery that , together with the SRAM, provides a non-volatile memory space  
25







1 The computer 202 runs an open platform operating system which supports  
2 multiple applications. Using an open platform operating system and an open  
3 computer system architecture, various software applications and hardware  
4 peripherals can be produced by independent vendors and subsequently installed by  
5 the vehicle user after purchase of the vehicle. This is advantageous in that the  
6 software applications do not need to be specially configured for uniquely designed  
7 embedded systems. In the illustrated example the open hardware architecture runs  
8 a multitasking operating system that employs a graphical user interface. A  
9 multitasking operating system allows simultaneous execution of multiple  
10 applications. One such operating system is the "Windows" brand of operating  
11 systems (e.g., the "Windows CE" operating system) sold by Microsoft  
12 Corporation of Redmond, Washington.

13 The computer 202 can include at least one storage drive which permits the  
14 vehicle user to download programs and data from a storage medium. In the  
15 illustrated implementation, the computer 202 has a CD ROM (or other optical)  
16 drive 220 which reads application-related CDs, as well as musical, video, game, or  
17 other types of entertainment CDs. The computer 202 may also include other  
18 storage devices, such as a magnetic disk drive, smart card reader, PCMCIA card  
19 sockets, a hard disk drive, flash memory, or a DVD ("digital video disk" or  
20 "digital versatile disk") drive. In the embodiment described below, non-volatile  
21 memory is provided in the form of flash memory. It is to be appreciated and  
22 understood that other storage devices can be used without departing from the spirit  
23 and scope of the described and claimed subject matter.

24 The storage drives are mounted in a base unit 228 of the housing 218. The  
25 base unit 228 is constructed and sized to be mounted in the dashboard. Optionally,

1 this base unit may be removable in the same fashion as a laptop computer and its  
2 associated docking station. This option allows the user to take the vehicle  
3 computer to his/her home or office to serve as his/her portable PC. The housing  
4 218 also has a faceplate 230 which is pivotally mounted to the front of the base  
5 unit 228 and may optionally be detachable. The faceplate can be rotated to permit  
6 easy and convenient access to the storage drives.

7 The computer 202 has a keypad 232 and a display 234 on the faceplate 230.  
8 The operating system executing on the computer 202 controls the faceplate  
9 components (keys, IrDA, display, knobs, touch screen, etc.), which, through the  
10 computer, can control the faceplate keys 232 and the faceplate display 234 as  
11 peripheral devices when the faceplate is attached to the base unit. Additionally,  
12 the computer 202 has a voice recognition device to permit the user to verbally  
13 enter commands in a hands-free, eyes-free environment. These voice commands  
14 can be used for controlling most operating modes of the vehicle computing  
15 platform. The computer 202 is also equipped with an IrDA (infrared developers  
16 association) transceiver port 236 mounted on the faceplate 230 to transmit and  
17 receive data and programs using infrared signals. The entire faceplate unit 230  
18 can behave as a multifunction peripheral to the computing platform.

19 The computer 202 can output visual data to the LCD 234 at the faceplate,  
20 or one or more display devices of the type shown in 204. In the exemplary  
21 illustration, display 234 is a back lit LCD and display 204 is a small flat panel  
22 display (e.g., 6.4" screen) that is movably mounted on a stand or yoke and  
23 remotely located from the computer. Additional display devices may also be  
24 added that are similar to display 204 or 234. Different types of display devices  
25 may also be added, such as a Heads Up Display (HUD).

1       The display 204 is fully adjustable to different viewing positions that can be  
2 seen by the driver or other passengers in the vehicle. The type of data displayed  
3 can range widely from word instructions concerning the vehicle's performance, to  
4 diagrammatic directions from a navigation system, to video movies for in-car  
5 entertainment. The display 204 can be equipped with a view indicator switch 238.  
6 This switch can indicate to the software whether the driver can view the display.  
7 Then, depending on whether the car is being driven or not, the software can  
8 determine which application content is appropriate to display on the display.  
9 When facing the driver, only information supportive and helpful to driving (e.g.,  
10 diagnostics, navigation directions) is displayed on the monitor, while distracting  
11 information (e.g., video movies, games) is blocked from display. In one  
12 implementation, the switch is an electrical cylindrical switch which closes when  
13 the display is capable of being viewed by the driver; thus, the software can sense  
14 the display position and only allow permitted information to be displayed.

15       In general, the vehicle computer system 200 can be used to integrate  
16 multiple vehicle-related systems onto one open platform hardware and software  
17 architecture. For instance, the vehicle computer system 200 can serve as a  
18 multimedia entertainment system, a navigation system, a communications system,  
19 a security system, and a diagnostics system. Moreover, the vehicle computer  
20 system 200 provides additional functionality traditionally associated with desk-top  
21 and laptop personal computers. For instance, vehicle computer system 200 can  
22 support word processing applications, spreadsheet applications, database  
23 applications, web browser applications, and appointment/schedule applications.  
24 Furthermore, the vehicle computer system 200 can be configured to operate as a  
25 server to other computing units in the vehicle to distribute games, video movies,

1 and the like to passengers. Furthermore, the system can include an internet  
2 connectivity engine 240 that is configured to establish connectivity with the  
3 Internet. This can be done in any suitable way. For example, the internet  
4 connectivity engine 240 can wirelessly establish connectivity with the Internet  
5 through known wireless techniques.

6 Information can be displayed on either display device 204 or display 234.  
7 The information can be provided by an application running on computer 202, or  
8 by a device external to computer 202, such as sensors 206 or via diagnostic  
9 interface 208, antenna 216, IrDA port 236, etc. Such information can also be  
10 provided to the computer via the internet connectivity engine 240. Information  
11 that can be displayed includes any type of data or control information.  
12 Additionally, information to be displayed can include a "caption" or "label" that  
13 describes the data. Examples of data that can be displayed include street  
14 addresses, phone numbers, and directions (e.g., "Turn Left At Light On Main  
15 Street"). Such data can be displayed either including a caption describing the data  
16 (e.g., "Address: 12345 Washington Street", where "Address:" is the caption  
17 portion of the information) or without a caption (e.g., "12345 Washington Street").  
18 Examples of control information include toolbars, menu options, and user-  
19 selectable on-screen regions (such as buttons), as well as instructions, headings,  
20 and other descriptive information.

21 In the discussion herein, the various embodiments are described in the  
22 general context of computer-executable instructions, such as program modules,  
23 being executed by one or more vehicle computers. Generally, program modules  
24 include routines, programs, objects, components, data structures, etc. that perform  
25 particular tasks or implement particular abstract data types.

The bus 356 represents one or more of any of several types of bus structures, including a memory bus or memory controller, a peripheral bus, an accelerated graphics port, and a processor or local bus using any of a variety of bus architectures. The system memory 354 can include read only memory (ROM) 358, dynamic random access memory (DRAM) 360, and non-volatile storage or flash memory 361. In addition, in accordance with the described embodiment, the system memory also includes a small amount of SRAM 363.

A portion of the operating system, such as kernel 362, contains the basic routines that help to transfer information between elements within computer 202, such as during start-up, can be stored in ROM 358 or flash memory 361.

A number of program modules can be stored in ROM 358 or flash memory 361 and transferred into DRAM 360. These include an operating system 364 and one or more application programs 366. A user may enter commands and information into computer 202 through various input devices, such as a keyboard (e.g., keypad 232 of Fig. 2), touchscreen, pointing device, microphone, joystick, game pad, satellite dish, scanner, or the like (not shown in Fig. 3). These and other input devices are coupled to the processing unit 352 through an input/output (I/O) interface 368 that is coupled to the bus 356. A display 204 or 234, or other type of display device, is also connected to the bus 356 via an interface, such as a video adapter(s) 370. Data to be displayed on display 204 or 234 is provided to adapter 370 by a display generator 372 of operating system 364. In addition to the





## Exemplary Memory System with Power Shutdown Capabilities

Fig. 4 shows an exemplary vehicle computer memory system 400 that is configured with power shut down capabilities in accordance with one described embodiment. System 400 comprises flash memory 402, DRAM 404, SRAM 406, a power control circuit 408, a power supply with low voltage detection circuit 410, a microprocessor 412, and a bus 414.

In the illustrated and described example, the vehicle's object store is maintained in flash memory 402. DRAM 404 is used to contain "read only" pages that are read in from flash memory 402. When the read only pages in DRAM 404 are written to or "dirtied", they are moved into SRAM 406 and maintained there. Low voltage detection circuit 410 is connected to the main voltage supply (i.e. the supply provided by the vehicle's battery). When the low voltage detection circuit 410 detects that the main supply voltage has been abruptly lost or has dipped below a defined voltage level, it generates an interrupt signal that is sent to the microprocessor 412. This notifies the microprocessor that there has been a power failure and enables the microprocessor to take the necessary steps that it needs to take in such an instance. This can include, for example, copying contents in DRAM 404 into SRAM 406. Power control circuit 408 is then activated just as regulated voltage VCC3V begins to drop out of regulation and ensures that a backup battery, in this case a small lithium cell, is incorporated in a circuit so that it can be used to power the SRAM. The power control circuit 408 is also configured to place SRAM 406 in a low power, high impedance state, while isolating the SRAM and the backup battery from the other system components. In the described embodiment, the SRAM is placed into the low power, high impedance state before the backup battery begins to provide power to the SRAM.

1 The illustrated system can also include a hold capacitor 414 that essentially  
2 enables the supply voltage to decay away as a function of the size of the hold  
3 capacitor. This provides an extra bit of time (~ 2-10 milliseconds) for copying any  
4 needed data from DRAM 404 to SRAM 406 or for setting any appropriate status  
5 information in SRAM 406.

6 In the illustrated circuit, the flash memory 402, DRAM 404, and SRAM  
7 406 are powered by a 3-volt power rail that is provided by the vehicle's battery  
8 through a suitable switcher power supply circuit (not specifically shown).

#### 9 10 Timing Considerations

11 Before continuing on with a discussion of one specific implementation of  
12 power control circuit 408 and power supply with low voltage detection circuit 410,  
13 the following discussion regarding certain timing considerations is provided so  
14 that the reader can more fully appreciate the purpose of the circuitry about to be  
15 described.

16 Fig. 5 shows a timing chart 500 that shows various voltages as a function of  
17 *time* in the event of an emergency or abrupt power shut down. In the automotive  
18 context, there is a main supply voltage of 12 volts that is provided by the vehicle's  
19 battery. This supply voltage is shown at 502. A switching power supply provides  
20 5-volt and 3-volt voltage rails that are used by various systems within the vehicle's  
21 computing device. For example, the 3-volt rail is used to power the flash memory  
22 402, DRAM 404 and the SRAM 406. These voltage rails are respectively shown  
23 at 504, 506. An interrupt signal is shown at 508 and constitutes the interrupt  
24 signal that is generated by the low voltage detection circuit 410 for microprocessor  
25 412.



1 incorporates and isolates the SRAM's backup battery so that it can power the  
2 SRAM.

3 Fig. 6 is a flow diagram that describes steps in a method in accordance with  
4 one described embodiment. In the illustrated example, the method is implemented  
5 in hardware, a specific example of which is given below.

6 Step 600 provides a vehicle computing device. Exemplary computing  
7 devices are described above. One particular type of vehicle computing device is  
8 an Auto PC device developed by Microsoft. It is to be appreciated and understood  
9 that the inventive techniques described below can be practiced in connection with  
10 any suitable vehicle computing device, with the Auto PC constituting but one  
11 exemplary device. Step 602 detects an abrupt power shut down condition. An  
12 abrupt power shut down condition can occur, for example, when a vehicle's  
13 battery is suddenly disconnected or otherwise unavailable for providing power to  
14 the vehicle's computing device. In the illustrated example, this step is  
15 implemented by a power supply with low voltage detection circuit such as circuit  
16 410 in Fig. 4. It is to be appreciated and understood that any suitable circuit can  
17 be used for implementing a power shutdown detection circuit. It is desirable for  
18 such a circuit to have characteristics that enable it to detect when a vehicle's main  
19 power supply is not available or will not be available in a short time. Step 604  
20 writes suitable data into SRAM after shutdown is detected by step 602. This step  
21 is shown in dash lines because it can be an optional step. The data that can be  
22 written into the SRAM can be data that is contained in the system's DRAM. Step  
23 606 places the SRAM in a low power, high impedance state for reasons that were  
24 mentioned above. In the illustrated and described embodiment, this step is  
25 accomplished by power control circuit 408. Any suitable circuit can be used to

1 place the SRAM in a low power, high impedance state, with but one exemplary  
2 specific circuitry being shown and described below.

3 Step 608 incorporates an SRAM backup battery into a circuit that provides  
4 power to the SRAM. The SRAM backup battery is desirably one that has a long  
5 lifetime so that the user need not worry about changing the battery. One  
6 exemplary lithium cell battery suitable for use in automotive environments is  
7 made by a company called Tadiran. In the illustrated and described embodiment,  
8 this step is implemented by the power control circuit 408, a specific example of  
9 which is shown and described below. Step 610 isolates the backup battery to  
10 power only the SRAM. Again, this step is implemented in the described  
11 embodiment by power control circuit 408.

### 12 13 **Exemplary Shutdown Detection Circuit**

14 In the illustrated and described embodiment, the low voltage detection  
15 circuit 410 is implemented as a comparator circuit within the power supply  
16 switcher circuit.

17 Fig. 7 shows one exemplary comparator circuit generally at 700 and  
18 comprises resistors 702, 704 connected as shown and having the indicated  
19 magnitudes. The circuit is connected to the main supply voltage indicated as  
20  $V_{\text{main\_bat}}$ . Hold capacitor 414 is provided as shown. A power supply switcher  
21 circuit 705 is provided and is connected to the main supply voltage to generate the  
22 5-volt and 3-volt power rails. A comparator element 706 is provided and includes  
23 a voltage reference input (the bottommost of the inputs) and an input derived from  
24 the voltage drop across resistor 704. When the input voltage to the comparator  
25 element drops below a predetermined nominal voltage, e.g. 1.65V, the comparator

1 circuit trips and generates an interrupt signal (referred to as an "NMI" interrupt for  
2 "Non Maskable Interrupt") on line 708 that is provided to the microprocessor.  
3 The microprocessor can then immediately copy any unsaved "dirty" pages (if any)  
4 or other needed information to SRAM and set any appropriate flags in the SRAM.  
5 Depending on the power supply used, when the input voltage drops to around 4V,  
6 the 3.3 power rail will begin to drop out of regulation which will cause the DRAM  
7 to lose its contents and the microprocessor will stop its functioning.

### 8 9 Exemplary Power Control Circuit

10 In the described embodiment, by the time the power rail to the SRAM has  
11 dropped out of regulation, the power control circuit has advantageously provided a  
12 backup battery for the SRAM, and well as isolated the SRAM and its backup  
13 battery from the other system components.

14 Fig. 8 shows an exemplary power control circuit 800. It is to be  
15 appreciated that the described power control circuit constitutes but one example of  
16 a suitable power control circuit. In the described embodiment, characteristics of  
17 the power control circuit are that it can accomplish one or more of the following  
18 functions in the event of an abrupt power loss or shut down: (1) smoothly  
19 transition to a backup power source for the SRAM, and (2) suitably isolate the  
20 backup power source from the other components in the system so that there is no  
21 current flow to the power plane.

22 Circuit 800 comprises a pair of series-connected diodes 802, 804 connected  
23 between the 5-volt power rail and the source of a P-channel field effect transistor  
24 (PFET) 808. A 2.2K resistor 806 is connected between (1) a node that is shared  
25 with both diode 804 and the source of PFET 808 and (2) the gate of PFET 808.

1 The gate of FET 808 is connected to the collector of a bipolar junction transistor  
2 (BJT) 810 whose base is connected to the 5-volt power rail through a 1K resistor  
3 812 and a 3.3V Zener diode 814. An N-channel field effect transistor (NFET) 816  
4 is provided and has its source connected to a chip select line for SRAM 818, and  
5 optionally SRAM 820. NFET 816 is essentially a switch on the chip select line  
6 the purpose of which will become evident below. The gate of NFET 816 is  
7 connected to the 5-volt power rail. A 47K pull up resistor 822 is connected  
8 between the drain of NFET 816 and the SRAM  $V_{cc}$  line.

9 A backup power source 824 is provided for the SRAM and comprises a  
10 3.6V lithium cell battery 826, a diode 828 connected to the battery 826, and a 470  
11 Ohm resistor 830 connected between the diode 828 and the SRAM  $V_{cc}$  line.

12 Circuit 800 works as follows: When the 5V power rail is within regulation,  
13 BJT 810 saturates thus creating a sufficient gate to source voltage to turn on PFET  
14 808. In this state, the SRAM 818 and optionally SRAM 820 are powered by  
15 approximately 3.6V (5V minus two diode drops) on the SRAM  $V_{cc}$  line.  
16 Advantageously, since this is the same voltage as backup battery 826, no current  
17 will flow out of the backup battery because diode 828 will not be forward-biased.  
18 Also, diode 828 will block any reverse current from attempting to flow back into  
19 or charge the non-rechargeable battery 826. It will be appreciated and understood  
20 by those of skill in the art that diode 828 could be replaced by a FET to ensure that  
21 no current flows through resistor 830 when the 5-volt power rail is within  
22 regulation.

23 When the 5-volt power rail drops well below regulation (i.e. less than about  
24 4V), Zener diode 814 ensures that BJT 810 turns off which, likewise, turns PFET  
25 808 off. With FET 808 turned off, the voltage on the SRAM  $V_{cc}$  line will begin to





1 To understand and appreciate concepts and the workings of an exemplary  
2 virtual memory system, consider Fig. 9.

3 Many modern computer systems, and indeed automotive computing  
4 systems of the type described above run multiple concurrent tasks or processes,  
5 each with its own address space. It would be expensive to dedicate a full  
6 complement of memory to each task, especially since many tasks use only a small  
7 part of their address spaces. Rather, virtual memory is used to give each process  
8 the appearance of a full address space. This allows a program to run on what  
9 appears to be a large, contiguous, physical-memory address space dedicated  
10 entirely to the program. In reality, however, the available physical memory in a  
11 virtual memory system is shared between multiple programs or processes. The  
12 memory that appears to be large and contiguous is actually smaller and fragmented  
13 between multiple programs. Each program accesses memory through virtual  
14 addresses, which are translated by special hardware and/or software to physical  
15 addresses.

16 Rather than attempting to maintain a mapping for each possible virtual  
17 address, virtual memory systems divide virtual and physical memory into blocks.  
18 In many systems, these blocks are fixed in size and are referred to as "pages". The  
19 addresses within an individual page can all have identical uppermost bits. Thus, a  
20 memory address is the concatenation of a page number, corresponding to the  
21 uppermost bits of the address, and a page offset, corresponding to the lower bits of  
22 the address.

23 A data structure is typically maintained in physical memory to translate  
24 from virtual page numbers to physical page frames. This data structure usually  
25 takes the form of a page table. A page table is indexed by virtual page number,

1 and it generally has a number of entries equal to the number of pages in the virtual  
2 address space. That is, the page table keeps track of every virtual address that has  
3 been allocated in the system and provides a mapping to a physical location.

4 Virtual-to-physical address translation can consume significant overhead,  
5 since every data access requires first accessing the page table to obtain a physical  
6 address and then accessing the data itself. To reduce address translation time,  
7 computers can use a specialized hardware cache that is dedicated to translations.  
8 This cache is referred to as a translation lookaside buffer (TLB). A TLB typically  
9 has a small, fixed number of entries. The entries correspond to recently translated  
10 virtual pages numbers or addresses. The TLB can be direct-mapped, set  
11 associative, or fully associative.

12 Fig. 9 shows an exemplary virtual memory system using a TLB 902 and a  
13 page table 906. A process typically generates a virtual address 900 comprising a  
14 virtual page number and a page offset. The page number portion of the virtual  
15 address is used to index a TLB 902. Assuming that the TLB contains an entry  
16 corresponding to the virtual page number (a situation referred to as a TLB "hit"),  
17 the TLB produces a physical page number. The page offset portion of virtual  
18 address 900 is concatenated with the physical page number from the TLB,  
19 resulting in a full physical address for accessing physical memory 904. If the  
20 correct entry is not present in TLB 902 (a situation referred to as a TLB "miss"),  
21 an initial reference is made to page table 906 to update TLB 902.

22 A TLB miss thus initiates its own memory reference that could in many  
23 cases be the source of another TLB miss, creating the potential for an endless loop.  
24 To prevent this, the page table is typically stored in an "unmapped" portion of  
25

1 physical memory that is addressed directly by its physical addresses rather than by  
2 virtual addresses.

3 In order to be able to page in object store pages on demand, to periodically  
4 write object store pages out to a non-volatile medium (such as SRAM) just before  
5 they are "dirtied" (i.e. written to), as well as to flush dirty pages to a non-volatile  
6 medium (such as flash), the location of each object store page is tracked.  
7 Furthermore, this location information is available across power-on boots.  
8 Because the above-described battery-backed SRAM never loses power, and  
9 because it is easily readable and writable, it is a desirable choice to maintain  
10 object store pages and page table information. The software portion the described  
11 embodiments can also be extended to use ordinary DRAM, instead of SRAM, and  
12 to store the page tracking table in FLASH. This, however, introduces more risk of  
13 data loss, and a guaranteed flash write time must be reserved with large enough  
14 capacitance, in order to allow the DRAM and page tracking table to be written to  
15 FLASH in the event of power loss.

16 Fig. 10 is a flow diagram that describes steps in a method in accordance  
17 with one embodiment. The method is desirably implemented in software in  
18 conjunction with an automotive computing device.

19 Step 1000 provides an automotive computing device. An exemplary  
20 automotive computing device is described above. Step 1002 provides an object  
21 store in non-volatile memory of the automotive computing device. In one  
22 implementation, the non-volatile memory comprises flash memory, although any  
23 suitable non-volatile memory will suffice. Step 1004 maintains one or more pages  
24 of the object store in one or more SRAMs of the computing device. Maintenance  
25 of the object store pages, as well as management thereof can advantageously take

1 place utilizing principles of virtual addressing. One specific virtual addressing  
2 system was described above, although any suitable virtual addressing system can  
3 be utilized.

4 Two primary goals of the described embodiments are to implement  
5 methods to reduce boot up time by demand paging the object store, and to  
6 implement an extremely fast shut down without data loss. The first goal is  
7 discussed in more detail below in a section entitled "Fast Boot". The second goal  
8 is met by performing a periodic flush of object store pages from DRAM to non-  
9 volatile memory (flash or battery-backed SRAM). Both these two goals share a  
10 core characteristic— that the object store can be paged between the various memory  
11 media in the system. In paging the object store it is useful for the software to track  
12 (1) which pages need to be paged, (2) when they need to be paged, and (3) where  
13 these pages are located. Each of these items is specifically discussed in the  
14 sections immediately below.

#### 15 16 Determining Which Pages Need To Be Paged In

17 In order to track which object store pages need to be paged in, pages that  
18 correspond to the object store are first identified. In the described embodiment,  
19 the object store is initially represented as a linked-list containing physical DRAM  
20 address pointers to pages of object store data. Software uses these linked-list  
21 addresses to determine the physical addresses of the object store. Furthermore,  
22 because these are physical addresses, rather than virtual addresses, they are not  
23 subject to change due to memory management tasks such as heap compaction. In  
24 accordance with the described embodiment, these physical addresses are placed  
25 within a table in battery-backed SRAM so that the object store can be tracked by



1 accesses. Additionally, when a TLB is full at the time of a TLB miss, at least one  
2 TLB entry is flushed to make room for a new one. Since a TLB is very small, a  
3 flush can occur frequently.

#### 4 5 Determining Where Pages Are Located

6 In general, the Operating System (OS) maintains all of the mappings  
7 between virtual and physical addresses in a software structure known as the page  
8 table which is described above in connection with Fig. 9. This OS structure is able  
9 to map every valid virtual address. A subset of these mappings is also kept in the  
10 CPU's hardware cache in the form of the TLB as mentioned above. The TLB  
11 contains the most recently translated virtual page numbers. In the illustrated and  
12 described embodiment, an additional table of object store pages referred to as the  
13 "SRAM table" or the "SRAM object store page table" (i.e. table 912) is also  
14 created and maintained in SRAM. Since the SRAM is battery-backed as described  
15 above, in the event of an abrupt power loss, all of the location data (i.e. virtual  
16 address to physical location mappings) for the object store pages are preserved and  
17 maintained in SRAM.

#### 18 19 SRAM Object Store Page Table

20 The following discussion describes a specific implementation of an SRAM  
21 Object Store Page Table and is not intended to limit application of the claimed  
22 subject matter. Accordingly, departures from the described implementation can be  
23 made without departing from the spirit and scope of the claimed subject matter.

24 In the illustrated and described embodiment, the object store is assumed to  
25 have a maximum size of 16 Mbytes. At this size, 4K object store page table







1       The SRAM information can be represented as a structure an example of  
2       which is given below:

```
3 typedef struct tagSramHeader{  
4     // if the table is valid, the following will be a specific magic number  
5     DWORD dwValidityFlag;  
6  
7     // last free block used for wear-leveling and compaction  
8     DWORD LastFreeFlashBlock : 5;  
9  
10    // index of flash block containing last page written to flash  
11    DWORD BlockforLastFlashPageWritten : 5;  
12  
13    // last page written to flash  
14    DWORD PageLastWritten : 7;  
15  
16    // number of object store pages in sram  
17    DWORD NumberSramObjectStorePages : 6;  
18  
19    // page number to write next object store page in sram  
20    DWORD IndexToWriteNextSramPage : 6  
21  
22    DWORD unused : 3;    // unused  
23  
24    // track free, erased, and erased blocks in flash  
25    DWORD FlashBlockInfo[256];  
26 } SramHeader;
```

### 27       SRAM Object Store Page Table Entry

28       Within the SRAM Object Store Page Table are individual *page table*  
29       *entries*. One specific exemplary SRAM Object Store Page Table Entry is shown in  
30       Fig. 11 at 1100. The “P” designations indicate bits that define a physical address,  
31       the “M” designations indicate bits that define a flash or SRAM address, the “L”  
32       designations indicate bits that indicate a location (i.e. DRAM, SRAM, or flash),  
33       the “D” designation indicates a bit that defines whether a page is dirty or not, the  
34       “N” designation indicates bits that define the number of times a page has been  
35       written to flash, and the “F” designation indicates bits that define a flash address.

On the very first-time boot of the automotive computing device (or after a reset which causes a cold boot), the system will go through a 5-10 second cold boot. Since this type of boot normally happens once in the lifetime of the device, it is assumed to be an acceptable specification, especially considering that it can be performed on the device before it is shipped to the customer.

In addition to general initialization, the described embodiment makes a copy of the object store in flash memory, and creates the corresponding SRAM Object Store Page Table in SRAM. This table is initialized with entries indicating that the pages are initially in flash memory with read-only access.

The described object store page table entries provide the information specified in the table below using the “L” bits.

L bits	Meaning
00	Page resides in DRAM, and there isn't a copy of it in flash yet.
01	Page resides in DRAM and there is a copy of it in flash.
10	Page resides in SRAM.
11	Page resides in flash.

#### P Bits

The “P” bits are used for tracking the start of the page relative to the starting physical address of DRAM. The offset of the start of the page relative to the starting physical address of DRAM corresponds to the fixed physical address offset at which the OS thinks the page is located in DRAM; because this is a fixed address, this address will not change when the page is moved from one physical medium to another.

With a 4 Kbytes page size,  $2^2 * 2^{10} = 2^{12}$  bytes can be accessed per page. Thus, the lower 12 bits of an address reference the offset within a page. Since the described table tracks starting page addresses, we can eliminate 12 bits of page offset from our 32-bit address.

Also, the upper six bits of our addresses are always the same for a specific physical medium; e.g. the upper six bits of addresses in flash are always 101000 binary. So, we can eliminate these upper 6 bits from the 32-bit address as well. This results in a  $32 - 12 - 6 = 14$  bit address; thus 14 bits are used to track the start of the page relative to the starting physical address of DRAM.

#### M Bits

The “M” bits are used for tracking the start of the page relative to the starting physical address within a physical medium. The offset of the start of the page relative to the starting physical address in SRAM, DRAM, or flash corresponds to the actual location of the page at any given time. Note that in the DRAM case, the P offset bits and the M offset bits are the same. (Note that “M” is a mnemonic for “moveable”.) When an object store page is moved from one physical medium to another, the “M” offset is changed to reflect the new physical location of the page. Using the same calculation as described in the P bits description above, 14 bits are necessary for this entry.

#### F Bits

The “F” bits are used to track the offset of the start of the page relative to the starting physical address in flash. The “F” bits are used in two scenarios: (1) When a read-only page is moved to DRAM, a copy of the page is still kept in flash

1 and its location in flash is tracked. The reason for tracking the page in both flash  
2 and in DRAM is that if power is lost, the read-only page in DRAM will be lost;  
3 (2) When a page is copied from SRAM to flash in order to make space for new  
4 pages in SRAM, the "F" bits are set to the starting address of the page in flash in  
5 order to track the page's new location in flash. Using the same calculation as  
6 described in the P bits description above, 14 bits are necessary for this entry.

### 7 8 D Bits

9 The "D" bit tracks whether a page is dirty (i.e. written to) or not. Only one  
10 bit is needed per object store page table entry to track whether the corresponding  
11 page is dirty or not.

### 12 13 N Bits

14 The "N" bits track the number of times a page has been written to flash. A  
15 count of the number of times the page has been written to flash can be used to  
16 determine which pages to flush back to flash when flushing becomes necessary.  
17 Tentatively, one byte per table entry is reserved.

### 18 19 Reserved Bits

20 The number of bits necessary for each page table entry may need to be  
21 modified in the future; we may also want to reserve additional bits for future  
22 expansion. Accordingly, the cross hatched bits correspond to those that are  
23 reserved for future use.  
24  
25

## OEM Adaptability

As noted above, the described embodiment provides an extensible platform upon which OEMs can build. Since the amount of physical memory present in a device is OEM specific, the data and bit fields described above in the "Object Store Page Table" and "Object Store Page Table Entries" sections can vary in size. Also, some parameters, such as page size, are CPU dependent. Because of this inherent variability, the information can be under OEM control.

As described below, access to these OEM-specific data can occur during "TLB Miss" and "Write Exception" handling. So called "hooks" can be provided to the corresponding OS exception handlers to call OEM exception handlers to access the OEM-specific data. Additionally, OS functions can be implemented to modify the OS page table and TLB; the OEM exception handlers can simply pass arguments, such as a new physical address or a page attribute, to these functions.

## Bootting With An Existing Configuration Scenario

As mentioned above, after the first power-on cold boot, all subsequent boots restore the minimum number of object store pages so that the operating system thinks it is warm bootting. Since every access to a page currently not represented in the TLB will cause a TLB miss exception, the system is able to "trap" every first attempted access to object store pages stored in flash and SRAM.

It is important to note that this particular described embodiment provides that the OS exception handlers for write access and TLB miss (for a read or a write) will not be modified, with the exception of having them also call an OEM exception handler (referred to as "hooking") and to handle return values from these hooked OEM handlers.



### **TLB Miss Error Exceptions**

Attempting to access (i.e. read and/or write) a page that is not currently mapped in the TLB results in a TLB Miss Error Exception. In the described embodiment, the OS TLB Miss Error Exception handler is modified to call an OEM exception handler to give it an opportunity to handle this exception.

If the page being accessed is part of the object store, the OEM handler is responsible for copying the page from flash into either DRAM or SRAM. If the exception occurred because of an attempted read access, the page will be copied to DRAM; if it occurred because of an attempted write access the page will be copied to SRAM. The reasoning behind this is that since a "read-only" page cannot be modified, it will never become dirty. In the event of a power loss, no changes to the page can therefore be lost. In addition, the SRAM Object Store Page Table is modified to reflect the new location of the page (i.e. L and M bits) as well as the dirty bit (D bit). Note that the dirty bit will be set to dirty on a TLB Miss Error Write Exception because the page will be dirty as soon as the write to it completes.

If the page is not part of the object store, the OEM exception handler can return to the OS exception handler with an indication that it did not process the exception. The OS exception handler can then process this exception normally – i.e. it will update the TLB with information for page corresponding to the attempted access.

### **Write Exceptions**

Attempting to write-access an object store page in flash, or a read-only object store page in DRAM will result in a write exception. The OS Write Exception handler can be modified to call an OEM exception handler to give it an

opportunity to handle this exception. If this page is part of the object store, this handler will be responsible for copying the page into SRAM for modification, as well as modifying the corresponding entries in the SRAM Object Store Page Table. These modifications include modifying the M bits to reference the page now in SRAM, the L bits to indicate the page now resides in SRAM, and the D bit to indicate that the page is dirty (i.e. it is about to be written to).

Also, the OEM exception handler is responsible for calling OS functions to modify the corresponding OS page table and TLB. These OS functions can be written within the OS layer and exported in the kernel. The corresponding OS page table and TLB entries are then modified to contain the page's new physical address, and to reflect that the page is now writeable.

## Summary of Handling Object Store Page Exceptions

Fig. 12 contains a table that generally summarizes the processing that takes place for handling object store page exceptions in accordance with the described embodiment. The explanation of the table is believed to be fairly straight forward and, for the sake of brevity, is not repeated here.

## Managing the SRAM

In the illustrated and described embodiment, the battery-backed SRAM is used, in the context of an automobile computing device, essentially as a write cache for the purpose of preserving dirty pages in the event of a power loss. The SRAM also includes state information which is saved in the SRAM Object Store Page Table. Accordingly, in the event of a power loss, all of the system's dirty pages as well as the locations of all of the Object Store pages are known. One





1 Step 1300 maintains multiple object store pages in SRAM. These pages  
2 can be provided into the SRAM as described above. Step 1302 periodically  
3 flushes one or more object store pages to non-volatile memory to make room for  
4 additional object store pages.

#### 6 Determining When To Flush SRAM To Flash

7 One approach to determining *when* to periodically flush SRAM pages to  
8 flash is to flush the pages at either low, normal, or very high thread priorities,  
9 based on how many free SRAM pages there currently are. For example, a low  
10 priority thread (such as an idle time thread) can flush the SRAM when there are  
11 less than  $x$  pages of free SRAM in which to write object store pages. A normal  
12 priority thread can flush the SRAM when there are less than  $y$  pages of free  
13 SRAM where  $y < x$ . Finally, a high priority thread can flush the SRAM when  
14 there are less than  $z$  pages (e.g. one page) of free SRAM, where  $z \ll y < x$ . The  
15 object is to minimize the writing done to flash and leave dirty pages in SRAM as  
16 long as possible. This prevents over usage problems with the flash memory and  
17 minimizes the amount of time spent copying pages from flash into ram and vice  
18 versa.

#### 20 Managing Flash Memory

21 An erased flash memory block consists of a series of '1's. Writing a 0 to  
22 any bit in the block requires the entire block to be erased. It takes a non-trivial  
23 amount of time to put the flash in erase mode and then to erase it. Also, the larger  
24 the block size you have, the longer it takes to erase a block.

1 Against this backdrop, two important issues are addressed when managing  
2 flash: compaction and wear leveling.

### 3 4 Compaction

5 In order to move one or more pages to flash, at least one block of erased  
6 flash needs to be available. This is because, unlike RAM, you cannot write to a  
7 portion of a block without erasing the entire block. If every block of flash had at  
8 least one page of object store data in it, you would need to erase one block before  
9 writing the new page. In the process of erasing the block, you would wipe out the  
10 contents of the page that was already in the block.

11 Fig. 14 shows a flash system 1400 with two blocks (Block 0 and Block 1)  
12 and a block size equal to  $4 * \text{page size}$ . Further, each block has one page of data  
13 in it. In order to write a new page to Block 0, the entire block first needs to be  
14 erased, meaning that the used page A would be lost. In order to prevent the page  
15 loss, at least one free block needs to be available to use for compaction. Thus, in  
16 accordance with the described embodiment, one block is reserved for compaction.  
17 For our example, page A, page B, and the new page are copied to an unused,  
18 erased block 2 as shown at 1402. After the copy, blocks 0 and 1 are erased to  
19 prepare for subsequent writes.

20 Fig. 15 is a flow diagram that describes steps in a compaction method in  
21 accordance with the described embodiment. The steps are desirably implemented  
22 in software, but can be implemented in any suitable hardware, software, firmware,  
23 or combination thereof.

24 Step 1500 provides flash memory arranged in blocks. Step 1502 reserves at  
25 least one block for compaction. Step 1504 copies one or more pages in one or

more other blocks to the reserved block. Step 1506 erases any blocks from which pages were copied to prepare the blocks for subsequent writes.

#### Wear-leveling

Flash memory can typically withstand 100,000 to 1 million writes before it fails. Depending on the amount of SRAM available, as well as the effectiveness of the algorithm used to determine which SRAM pages to flush to flash, the SRAM may or may not need to be wear-leveled.

Wear leveling requires using a different free block for each successive compaction. This is because if the same block in flash is always used as the free block, eventually this block will wear out. For the 0<sup>th</sup> compaction, block 0 can be used as the free block in which to copy pages; for the n<sup>th</sup> compaction, block n could be used. Accordingly, a different measure can be used to determine which block to use.

A variable in SRAM can track the current free block and the offset of the last written page since this value needs to persist across power-off boots. In addition, the next page written to flash can be written to a block at a physical address always greater than address of the previous page write.

As an example, consider Fig. 16 which shows a flash system 1600 with 3 blocks of 4 pages each. Initially, block 0 is reserved as the free block for compaction. Blocks 1 and 2 contain 4 used pages each. However, after some time later, the second page of each block is paged to SRAM. The next page write to flash cannot be done to blocks 1 and 2 because these blocks need to be erased before the next write (see the "Before Compaction" diagram on the left). In order to prepare for the next page write, flash memory needs to be compacted. Since

1 block 0 is the free block for compaction, pages will be written to it after block 0 is  
2 erased (or it may have been previously erased during idle time). Since we are also  
3 wear leveling while compacting, the goal is to reserve block 1 (the next  
4 consecutive block after block 0) as our next free block. This means that we must  
5 move all used pages currently in block 1 to block 0 (See the "After Compaction"  
6 diagram on the right).

7 After all of the pages in block 1 are moved to block 0, block 1 can be  
8 erased so that it can be used as the next block to which pages are moved.

### 9 10 **Fast Boot**

11 One design goal in the presently-described automotive computing device is  
12 to be able to boot the device from a zero power state, i.e. get the device up and  
13 running, as fast as possible. In the past, one solution to the zero power scenario  
14 was to take the entire object store and maintain it in flash memory. On boot up,  
15 the entire object store would then be read into DRAM. This process involved  
16 copying every single page in the object store, whether the page was needed or not,  
17 and could take on the order of 5 seconds or more. In accordance with the  
18 described embodiment, the only pages of the object store that are copied into  
19 RAM are pages that are actually needed for boot up. That is, the necessary pages  
20 are paged into RAM from flash memory as they are needed during a boot from a  
21 zero power state. These necessary pages can include, without limitation, pages  
22 that are used by the file system during its initialization. These can include pages  
23 that contain the file system's structural data, FAT tables, and the like. As  
24 subsequent pages from the object store are needed, they are paged in. Thus, in the  
25 "fast boot" scenario, pages from flash memory are copied into RAM as they are

1 being used. This reduces the number of pages that are read in on boot up from (a)  
2 all of the pages in the entire object store to (b) just the pages that are needed for  
3 booting. It will be appreciated that the specific pages that are needed by any  
4 specific system are not particularly relevant to this discussion, as such pages can  
5 change as between systems.

6 Fig. 17 is a flow diagram that describes steps in a fast boot method in  
7 accordance with the described embodiment. In the illustrated example, the  
8 method is implemented in software that is executing on an automotive computing  
9 device such as the devices described above. Step 1700 maintains a copy of an  
10 object store in non-volatile storage. The non-volatile storage can be any suitable  
11 non-volatile storage that comprises part of the vehicle's computing device. In one  
12 implementation, the non-volatile storage comprises flash memory. The object  
13 store copy is maintained in the non-volatile storage from a previous boot of the  
14 system. Step 1702 initiates a boot sequence of the automotive computing device.  
15 Such sequence is initiated when, for example, power is returned or supplied to the  
16 system, i.e. when a vehicle carrying the computing device is started or has power  
17 restored. Step 1704 pages, during the boot sequence, only predetermined object  
18 store pages from the non-volatile storage into RAM. Desirably, this results in less  
19 than all of the object store pages being paged in during boot up. This enables the  
20 computing device to boot quickly. Step 1706 determines, after the boot sequence,  
21 whether any additional object store pages are needed. If any pages are needed,  
22 step 1708 pages them in as described above.

23 In one implementation that utilizes an SRAM as described above, there is  
24 software code that recognizes whether the SRAM object store page table is valid,  
25 e.g. the code uses a signature in a known manner. The SRAM object store page

1 table contains information that describes what has been done with all of the  
2 DRAM file system pages. That is, with respect to all of the object store pages, and  
3 particularly those that were contained in DRAM before the last power down, the  
4 SRAM object store page table contains information that describes these pages'  
5 locations. When a boot up is initiated, software code processes the SRAM object  
6 store page table and marks all of the object store pages that were indicated as  
7 residing in DRAM as not being in DRAM any longer. Note that these pages are  
8 still in the flash memory because presumably, they were "read only" pages copies  
9 of which are maintained in flash memory.

10 When the operating system tries to reference a page that it believes should  
11 be in DRAM, a TLB miss exception is generated. Recall that TLB miss  
12 exceptions are generated the first time a page is accessed since the TLB does not  
13 yet contain a mapping between the virtual address of the page and the physical  
14 address at which the page is actually located. In accordance with the described  
15 embodiment, the TLB miss exception is intercepted before the operating system  
16 has a chance to process it. Software code then takes the virtual address of the page  
17 and uses it as a key into the OS page table. This is done to fetch the physical  
18 address (in RAM) at which the operating system believes the page is located. The  
19 software code then compares this physical address with the addresses in the  
20 SRAM object store page table to ascertain whether the page comprises part of the  
21 object store or not. If the page is part of the object store, the code ascertains the  
22 page's actual location by using the SRAM object store page table. Recall from the  
23 discussion above that various bits are used to track the current location of a page  
24 as being in either SRAM, DRAM or flash. If the page is currently in SRAM, an  
25 adjustment is made to the OS page table to change the page's physical DRAM





